Managing Data in Computational Edge Clouds

Mohan, Nitinder

ACM
2017-08-09

Mohan , N , Zhou , P , Govindaraj , K & Kangasharju , J 2017 , Managing Data in Computational Edge Clouds . in MECOMM '17 Proceedings of the Workshop on Mobile Edge Communications . ACM , New York , pp. 19-24 , Workshop on Mobile Edge Communications , Los Angeles , California , United States , 21/08/2017 . https://doi.org/10.1145/3098208.3098212

http://hdl.handle.net/10138/307235
https://doi.org/10.1145/3098208.3098212

unspecified
acceptedVersion

Managing Data in Computational Edge Clouds

Nitinder Mohan[†] Pengyuan Zhou[†] Keerthana Govindaraj[‡] Jussi Kangasharju[†]
[†]University of Helsinki, Finland [‡]Robert Bosch GmbH, Germany

ABSTRACT

Edge clouds handle data and computations closer to its source and users. Applications like industrial automation, bring new challenges and require solutions tailored for computation-centric edge cloud networks. In this paper we build on existing edge and fog computing models and develop a solution to predict and store data in edge resource caches for upcoming computations. Our solution is based on grouping caches according to the workloads they serve. We further develop methods for populating the caches and ensuring the coherence of the cached data. We evaluate the performance of our grouping mechanisms and show that they bring significant performance gains, both in terms of network traffic and access latency.

CCS CONCEPTS

• **Theory of computation** → *Caching and paging algorithms*; • **Computer systems organization** → *Cloud computing*;

KEYWORDS

Edge cloud, fog cloud, edge caching, cache groups

ACM Reference format:
Nitinder Mohan, Pengyuan Zhou, Keerthana Govindaraj, and Jussi Kangasharju. 2017. Managing Data in Computational Edge Clouds. *In Proceedings of MECOMM '17, Los Angeles, CA, USA, August 21 2017*, 6 pages. http://dx.doi.org/10.1145/3098208.3098212

1 INTRODUCTION

Edge clouds are a new and attractive way of handling large-scale data analysis closer to the clients at the network edge. Edge clouds offer several benefits including decreased latency for clients, reduced network traffic, and better handling of information that is of local interest. Different models for edge computing have been proposed [3, 11] and we follow our proposed Edge-Fog cloud model for this work [3]. *Edge-Fog cloud follows a three-tier hierarchy which consists of lower-powered edge devices closest to the users, fog devices with more computational power, and a central data store for permanent archival of data.*

While the data store provides permanence, solely relying on it for storing computational data adds considerable delay for fetching data to edge resources. Hence, caching data at the edge seems to be the obvious answer as it yields several benefits [4, 6] as well. However, we need to address additional challenges on how to manage, discover, and use the data cached at the edge. Existing solutions [4] propose CDN-like models which is not appropriate for a computation-first network as necessary for edge cloud application scenarios. When compared to CDNs, data in Edge-Fog cloud has shorter temporal relevance and receives more frequent updates.

In this paper, we propose an efficient edge caching mechanism leveraging the edge and fog resource caches to predict and store data required for upcoming computations. Our target applications are in industrial environment, particularly in factory automation and collaborative robotics. This paper makes the following contributions. First, we define a model and methods for cache grouping in an Edge-Fog cloud. Second, we develop mechanisms for ensuring coherency of cached data. Third, we evaluate the performance of our grouping solutions in a simulated Edge-Fog environment and show that grouping based on workload type brings significant performance gains such as reduced network traffic and access latency. We also discuss the optimal size of such resource groups and importance of workloads in the system.

The paper is structured as follows. Section 2 presents the application scenarios and Section 3 reviews related work. We present our solution in Section 4 and discuss communication matters in Section 5. Section 6 presents our evaluation. Finally, conclusion of the work has been drawn in section 7.

2 APPLICATION SCENARIOS

Recent studies predict close to 50.1 billion IoT devices will be connected over the Internet by 2020 [1]. Data generated by these devices will require time-critical processing and management to support fault resistant applications such as augmented reality, autonomous driving, video analytics etc.

Automation also extends to factories and acts as a driving force behind next generation manufacturing industries. The production system needs to be made faster, flexible, and

MECOMM '17, August 21, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5052-5/17/08…$15.00
https://doi.org/http://dx.doi.org/10.1145/3098208.3098212

cost-efficient to cope with increasing demands. Factories can achieve low latency computations by allocating tasks on edge clouds. Edge nodes can process data from automated tools and sensors to be reconfigured based on the task requirements.

However, factory tasks rely heavily on the availability of required data at compute time. Specification of end products can significantly vary, requiring on-the-fly calibration of the tools for each workpiece. Such recalibration information must be cached at edge nodes to ease subsequent task processing. For example, a machine meant for drilling holes must be able to change its settings for the next workpiece or switch to a different task altogether such as driving screws. Further, industries have started collaborative robots such as Bosch APAS [2] that work in tandem with human operators. Such robots need time-critical processing to create a safety zone for its operator while executing future demands. Relevant warning and sensor information must be cached at edge nodes to achieve sufficiently low processing requirements.

Autonomous transportation systems within a factory also impose similar requirements by requiring optimal and updated routes in extremely low time bounds. Required map data must be pre-cached and updated to all vehicles with critical information such as accidents or path congestion within a reasonable time. Augmented Reality glasses can assist operators in a continuously varying production environment by performing markerless object recognition and accurate tracking in a factory. This requires comparing real-world objects with pre-created 3D models stored in a remote data store. The fluidity and QoE of these devices significantly rely on bounding data retrieval delay within human reaction time.

Apart from the applications mentioned above, many other Cyber-Physical-Production-Systems (CPS) data in edge clouds needs inter-operation and communication which can only be achieved by efficient caching and data sharing within cloud resources.

3 RELATED WORK

Several edge cloud models such as Cloudlets [7], nano data centers [8, 12], community clouds [9], CISCO Fog [11] have been proposed to perform computation tasks at the edge of the network. However, these models assume that the required data for computation at a node is available in local caches of edge resources. That does not always hold as edge cache hit ratio is heavily dependent on the deployed workload type [13]. Further, they do not consider the impact of fetching the data from a data center into the edge cache and the subsequently added delay on workload computation.

Content Delivery Network (CDN) models aim to distribute content to end users via distributed servers and edge cache hierarchies [14, 15]. Edge caching is also a major motivation behind the design of 5G technology [16]. Exploitation of in-network caching to enable more efficient content distribution serves as a motivation behind information-centric networking (ICN) research. However, both CDN and ICN assume a single publisher/owner of the data which holds the rights for updating that content in future [6]. These networks follow a *push-based* approach wherein the owner pushes its data update to the central repository which broadcasts that update to every edge cache hosting that data. However, such an approach is inefficient for computational edge caches as the local copy of shared data can frequently be updated simultaneously by several edge resources. Notifying the central database of every update can lead to a severe network congestion and does not scale.

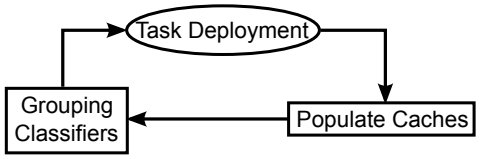


Figure 1: Edge-Fog cloud caching algorithm

Cooperative caches groups at the edge of the network have been proposed to avoid recurrent updates to the central database. Ramaswamy et al. [17] clustered edge caches into cooperative groups based on their proximity to other caches and the origin server. Our proposed cache grouping technique significantly differs from their approach. The authors cluster resources based on their network distances to other resources whereas we consider locally cached content as clustering classifier. Using network distance for clustering in Edge-Fog cloud would significantly lower the efficiency of task deployments which also considers the processing power of Edge/Fog resources. Furthermore, Ramaswamy et al. they do not consider parallel updates within cache groups and therefore do not propose a coherence model to mitigate invalid simultaneous updates.

4 RESOURCE CACHE GROUPING

Resources in Edge-Fog cloud request data from data store into their local cache according to end application requirements. Task deployment algorithms for Edge-Fog cloud, such as LPCF, designates a set of resources (Edge and Fog alike) for available tasks on to achieve least processing and network cost involved [3]. However, in a system with varying workloads, such a deployment reduces cache re-usability as same set of resources might be allocated to workloads with different data requirements in subsequent computations. This further leads to higher cache misses and higher network latency for fetching required data from data store into local cache thereby delaying the overall computation.

We consider Edge-Fog compute resources as collection of edge caches represented by $RC = RC_0, RC_1, ..., RC_{n-1}$. Every

edge cache stores data D_i as per its application task requirement. Deployed tasks in cloud can be classified into workloads W of k types. A resource computing workload W_k will require data classified according to that workload D_i^k in their cache. We propose a cache grouping algorithm which aims to cluster caches into cache groups $CG = CG_0, CG_1, .., CG_{K-1}$ based on their local cached content classification. Cache groups are not disjoint as resource RC_x can be part of more than one $\{CG\}$ if it has cached data for different workloads. The size of the group, $|CG|$ denotes the number of members of that group. The caches within a group can maximize their cache hit ratios and lower network delays by sharing data with other group members in future deployments.

4.1 Grouping Algorithm

We propose a three-step iterative cache grouping algorithm which builds up on the available task deployment algorithms. The algorithm is shown in Figure 1.

At time $t = 0$, none of the resources in Edge-Fog cloud have any tasks assigned to them and thus have no data in their local cache. At computation arrival time t_c, task deployment algorithm deploys an application task on a set of $\{RC\}_i$ resources which then retrieve the required data from data store into their local caches (phase "Populate"). As all resources involved in the computation belong to same task, they cache same or related content in their local cache. The computation is classified as part of workload W and all resources $\{RC\}_i$ are grouped in a single abstract cluster $\{CG\}_i$. As several parallel computations are deployed on the cloud, at time $t = n$ computations deployed are classified in W_k workloads which form CG_k resource cache groups.

In the next iteration, the task deployment algorithm prioritizes deploying next application task on a cache group which handled that workload in the previous cycle. This enhances the cache re-usability in resources belonging to that group. In case the resources in a group are non-ideal for a task deployment, other resources (independent or other group members) are considered for computation. Size of a cache group increases as more resources compute tasks and cache content for that particular workload. As a result, a cache can be member of more than one group based on its cached content classification. Figure 2 shows a snapshot of Edge-Fog cloud resources which have been grouped in two cache groups.

Within each cache group a resource is assigned as a *leader* (depicted with crown in the figure) which acts as a representative and communication backbone of the group. The leader is responsible for maintaining a coherent copy of data within a group and to enable content sharing among group members. The leader is required to have a consistent connection with all of its group members, exploiting the Fog resources in the model if needed. A distributed election algorithm can be used to elect a group member as a leader. The group leader can also replicate its local data structures on a secondary node which acts as a backup leader to ensure consistency despite failures.

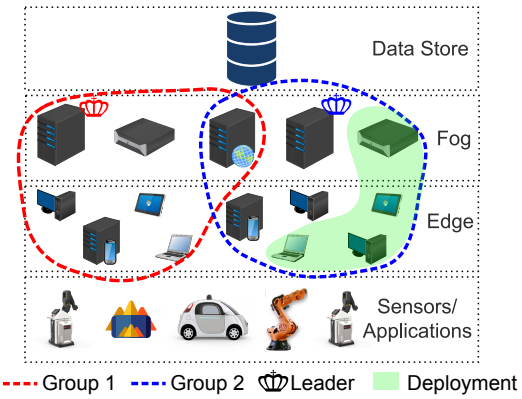
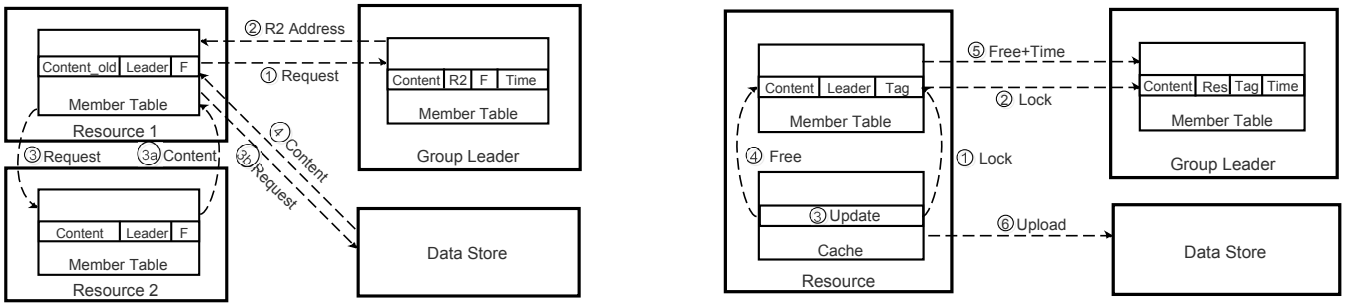


Figure 2: Edge-Fog cloud grouped resources

4.2 Grouping Classifier

Our cache grouping algorithm relies on data classification in the Edge-Fog cloud. As mentioned in Section 2, different applications require different sets of data at different times for their compute tasks. Data can be clustered according to their similarities which can be exploited to form dedicated cache groups for a data type. Several classification metrics can be used to achieve such groups.

1) **Location**: Data can be classified according to its location of generation or usage. For example, Augmented Reality headsets require 3D model and augmentation based computations only on objects within their field of vision.
2) **Relevance**: Data sharing attributes for re-configuration is a good classifier. Collaborative robots submitting production tasks of mobile and laptop cases can be grouped under `casing` attribute.
3) **Pending tasks**: Factory environments are flexible and dynamic wherein tasks are not bound to robots; instead the robots choose from a pool of pending tasks.
4) **Time**: Data generated by sensors are relevant to the end resources only for a particular period which can range from a few hours to a couple of days.
5) **Personalized settings**: Collaborative robots work with human operators who can configure them according to their specific needs.
6) **Routes/maps**: Autonomous robots are often mobile and require constant computation of optimal paths devoid of hinderances. Continually updated maps must be made available at frequent time intervals.
7) **Warning signals**: Data relevant to the safety have higher importance over other information and need to be made available to all the devices until categorized as invalid.



(a) Data retrieval within cache group

(b) Data update within a cache group

Figure 3: Communication Model

8) **Sensor information**: Actuators respond and regulate themselves based only on particular sensor data.

Above are examples of possible classifiers relevant for a factory automation scenario. However, for optimal operation, one or more groups need to be considered at the same time and groups must be weighted differently for each category of end devices, as not all the information is equally relevant.

5 GROUP COMMUNICATION

Resources clustered in a cache group need to communicate with their group members to share updated data efficiently. However, an effective communication technique needs to fulfill the following objectives for efficient operation.

1) Reduce *unnecessary network traffic* by exchanging data between resources only when needed.
2) Ensure *consistent copies* of data by avoiding computation on stale copies.

Considering above objectives, we propose a communication model which introduces a set of tabular data structures attached to resource's cache. We further present a low overhead message flow model to update and retrieve shared data within a group. Our model ensures *causal coherence* on shared data and is highly inspired by *directory cache coherence algorithm* [10] for networked processing systems.

5.1 Cache Data Structures

We now define data structures deployed with resource caches to assist data sharing within the group. The data structures provide content information to data stored in resource local cache. We deploy tables at three entities in the system: member resource, group leader and the Data Store.

Group Member Table: Every resource in Edge-Fog cloud associates itself with a content group and maintains a table to help content sharing with other group members. Members maintain a local cache table with the following entries:

a) **Data Name**: URI of a content cached in local cache
b) **Leader Address** Address of the group leader responsible for synchronization of that content
c) **Tag** State of data within its local cache. If data is currently used for computation, tag entry is *locked* otherwise *free*

The group member table maps locally cached data to groups based on their respective *leader address. Tag* is required for providing coherence within the group and is explained in further sections of this paper.

Group Leader Table: The group leader acts as a communication gateway between members of the group. To maintain the current state of data flowing within the group, the leader maintains a group leader table with following information:

a) **Data Name**: URI of the content cached within the group resource caches
b) **Tag**: Maps to content tag in member resource cache
c) **Resource Address**: Address of resource which updated the data. The resource also acts as host of that content within a group
d) **Timestamp**: Time at which resource notified the leader after updating the content

The *group leader table* helps ensure that the leader has addresses of host resources and the content state is synchronized between a resource and group leader.

Data Store Table: Data Store is the central repository and backup of cached content in Edge-Fog cloud. Resources update their content in the Data Store after every computation. The Data Store table has the following entries:

a) **Data Name**: URI of data stored in Data Store
b) **Classifier Type**: Classification property used for mapping content to a cache group
c) **Leader Address**: Address of group leader handling synchronization of that content

5.2 Communication Flow

We use the information stored in data structures described in previous section for ensuring that content gets updated properly. We use a *pull-based* model within a group which

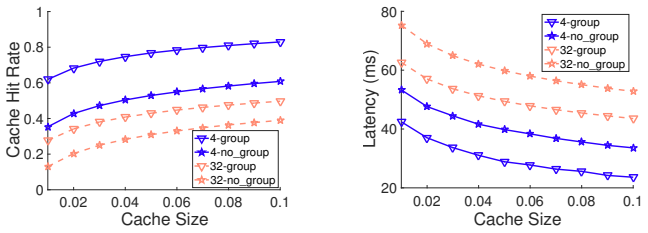
limits the number of messages in the network while ensuring data consistency. We assume that the data store acts as a central roll-back in case of failures in the system. To ensure this, the resources upload their updated data to the Data Store after each successful computation. As data upload happens *in-parallel* to task computation and data retrieval, it does not impact the computation time in the cloud. We further assume that messages in the system are not lost or corrupted in transmission.

Retrieving Content: A naive way to update cached content is to retrieve it from the data store. However, the main objective of forming cache groups in Edge-Fog is to assist content sharing amongst the computing resources. Retrieving content within a cache group must also preserve the coherence among multiple content copies in other edges of the network.

The communication model for retrieving content within a group is shown in Figure 3a. The model ensures *causal coherence* by sharing only last known updated content within the group. The group leader acts as information dissemination entity for the group. Every resource requests updated copy of content before initiating computation on its locally cached copy. The request is sent to the group leader which checks its table and returns the address of the node which last updated the content. The requesting node directly queries for the content from last-updater node. In case the node is alive and has the data in its local cache, it sends the content back to the requester. Otherwise, request is sent to the data store to retrieve backup copy.

Updating content within a group: Content in an Edge-Fog cloud resource group is continuously updated after each successful computation. However, to mitigate invalid results, resources must always compute on the most relevant copy of required data. A naive approach is to push the updated data to all members of the group which house copy of that data in their local cache. However, this leads to unnecessary flooding in the network which impacts network latency.

Instead, we employ a pull-based, step-wise checkpoint approach for handling updates. The message flow is shown in Figure 3b. Before computing on a locally cached copy of content, a member resource inquires its group leader for any updates on that copy of data. In case data has been updated by another member, the requesting node retrieves the latest copy by following the model described above. After successful retrieval, the updating node marks its *"update-in-progress"* by tagging its locally cached content as *locked* and asks the group leader to do the same. After a successful update, the resource un-tags its cached content as *free* and notifies the leader of the completed update. The leader, in turn, marks the particular content as *valid* along with the timestamp of the operation. This operation prevents any other resource to retrieve data under update. Finally, the resource updates the Data Store with the computed data.



(a) Effect on cache hit rate **(b) Effect on latency**

Figure 4: Variable grouped resource cache size analysis

6 EVALUATION

We implemented our system in Icarus [5] on a topology of 320 Edge and Fog resources and a central data store which stores all content in network. We clustered resource caches into evenly divided groups of various sizes. A Fog resource in each group is assigned the task of group leader. Network delays were modeled according to [3, 18].

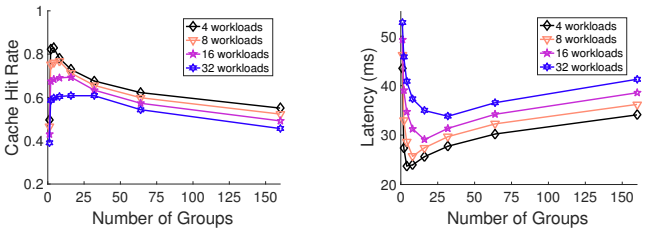
A workload is defined as a request distribution following a power law distribution. We generate requests for $96 * 10^4$ content items divided in upto 32 different workloads. A resource can store maximum of 10% of overall contents in the network in their local cache. Caches utilize Least Recently Used (LRU) cache replacement policy for swapping their cached contents. Cache retrieval and updates follow the communication models described in section 5.2 coupled with ideal Nearest Replica Routing (iNRR) algorithm [19].

6.1 Grouped vs. Non-Grouped

We first compare the effect of grouping on system performance. Figure 4 shows the cache hit rate and network latency after grouping resource caches. For optimal comparison, we cluster caches into same number of groups as the number of workloads deployed to ensure 1:1 mapping (see below).

Figure 4a shows cache hit rate after grouping. For both 4 and 32 workloads, grouping almost *doubles* the overall cache hit rate. Similarly, figure 4b shows that the latency of fetching the content decreases by up to 45% after grouping. The results clearly indicate that our grouping strategy significantly improves content management in edge clouds.

Effect of Cache Size: The results also show that cache grouping is most effective when cache sizes are small. As we increase the cache size of computing resources, the cache hit and latency gains slightly diminish. The reason behind this decrease is the overall fraction of the content that can be cached in the system. When resource cache has limited size, the amount of content that it can cache is low and grouping several resources in cache groups increases the probability



(a) Effect of group sizes on cache hits

(b) Effect of group sizes on latency

Figure 5: Resource cache grouping analysis for various workload sizes

of serving locally cached content. On the other hand, as cache size increases, it can cache more content in the network which increases cache hits even in disparately placed resources, closing the gap between non-grouped caching and grouped strategies.

Effect of workload sizes: Figure 4 also shows a correlation between performance gain and workload size. Both cache hit ratio and latency performs much better for lower workload sizes. *A workload is modeled as uniform distribution of similar content requests.* Lower workload sizes depict that overall content requests are very similar and thus can be satisfied by local caches of resources. As the content requests start being more unique, resources undergo more cache misses thereby inducing latency while satisfying a request.

6.2 Variable Group Size Analysis

We analyze the effect of group size $|CG|$ and number of groups $\{CG\}_k$ on system performance. Figure 5a shows the impact of variable group sizes on cache hits in the system. We compare the performance for several workload sizes and analyze how they affect the cache performance.

For all workload sizes, cache hit rate is is observed to be low as due to lack of cache grouping, content requests are handled by all resources in the cloud. As resources grouping in the system increases, the request types start converging on a single cache group. Maximum hit rates are achieved when *number of cache groups equal the number of workloads.* This 1:1 mapping ensures that each workload is being handled by a dedicated cache group, eliminating any overlap. Increasing the number of cache groups more than available workloads leads to overlap and duplication of content which reduces the cache hit performance of the network.

A similar trend is also seen for latency in figure 5b. The content retrieval latency is inversely proportional to cache hit rate of the system. As the cache hit rate increase, the latency to retrieve content decreases and reaches a global minimum at 1:1 deployment of workload and groups.

7 CONCLUSION

We have presented a grouping strategy for managing content in edge cloud caches. Our grouping is based on classifying content based on their workloads and caching related content on same caches. Our communication model provides causal data coherence while enabling parallel updates. We have evaluated our approach via simulations and have shown that grouping based on workload type significantly improves system performance in edge clouds through reduced network traffic and access latency. Our results show that the optimal number of groups is the number of workloads on the system.

REFERENCES

[1] Broadband by the numbers,'https://www.ncta.com/broadband-by-the-numbers', accessed: 2015-04-22.
[2] Robert Bosch GmbH Bosch APAS description. http://www.bosch-apas.com/en/apas/produkte/assistant/apas_assistant_3.html. Accessed: 2017-02-09.
[3] Mohan, N., and Kangasharju, J. Edge-Fog cloud: A distributed cloud for Internet of Things computations. In *Cloudification of the Internet of Things* (November 2016), IEEE.
[4] Ramaswamy, L., Liu, L., and Iyengar, A. Cache clouds: Cooperative caching of dynamic documents in edge networks. In *ICDCS* (2005), IEEE, pp. 229–238.
[5] Saino, L., et al. Icarus: A caching simulator for information centric networking (icn). In *ICST SimuTools* (2014).
[6] Zhang, G., et al. Caching in information centric networking: A survey. *Computer Networks 57*, 16 (2013), 3128 – 3141.
[7] Verbelen, T., et al. Cloudlets: Bringing the cloud to the mobile user. In *Proceedings of ACM workshop on Mobile cloud computing and services* (2012).
[8] Shi, C., et al. Serendipity: enabling remote computing among intermittently connected mobile devices. In *Proceedings of ACM MobiHoc* (2012).
[9] Li, Yujin and Wang, Wenye Can mobile cloudlets support mobile applications? In *Infocom proceedings ieee* (2014).
[10] Agarwal, A. et al. An Evaluation of Directory Schemes for Cache Coherence In *SIGARCH Comput. Archit. News* (1988).
[11] CISCO Cisco fog computing solutions: Unleash the power ofthe internet of things (whitepaper).
[12] Hong, K., et al. Mobile fog: A programming model for large-scale applications on the internet of things In *Proceedings of ACM SIGCOMM workshop on Mobile cloud computing* (2013).
[13] Sitaraman, R. K., et al. Overlay networks: An akamai perspective In *Advanced Content Delivery, Streaming, and Cloud Services* (2014).
[14] Pierre, G., Van Steen, M. Globule: a collaborative content delivery network In *IEEE Communications Magazine* (2006).
[15] W. Zhu and C. Luo and J. Wang and S. Li Multimedia Cloud Computing In *IEEE Signal Processing Magazine* (2011).
[16] Wang, X., et al. Cache in the air: exploiting content caching and delivery techniques for 5G systems In *IEEE Communications Magazine* (2014).
[17] Ramaswamy, L., et al. Efficient formation of edge cache groups for dynamic content delivery In *Proceedings of IEEE ICDCS* (2006).
[18] Rajahalme, J., et al. On name-based inter-domain routing In *Elsevier Journal on Computer Networks* (2011).
[19] Rossini, G. and Rossi, D. Coupling caching and forwarding: Benefits, analysis, and implementation In *Proceedings of ACM ICN* (2014).